US-PAT-NO: 6260157

DOCUMENT-IDENTIFIER: US 6260157 B1

TITLE: Patching of a read only memory

DATE-ISSUED: July 10, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Schurecht; Kurt	St. Charles	IL	60174	N/A
Bennett; Steven W.	Lake Zurich	IL	60047	N/A
Porter; James L.	Wauconda	IL	60084	N/A

APPL-NO: 09/ 250888

DATE FILED: February 16, 1999

INT-CL: [07] G06F009/26

US-CL-ISSUED: 714/8, 711/202 , 714/7

US-CL-CURRENT: 714/8, 711/202 , 714/7

FIELD-OF-SEARCH: 712/200; 712/32 ; 717/4 ; 714/5 ; 714/38 ; 714/8 ; 714/7 ; 711/108 ; 711/5 ; 711/202 ; 713/3

REF-CITED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
4028678	June 1977	Moran	340/172.5
4400798	August 1983	**Francis** et al.	365/174
4542453	September 1985	Patrick et al.	364/200

4751703 371/10	June 1988 N/A	**Picon** et al. N/A
4802119 364/900	January 1989 **N/A**	Heene et al. N/A
4831517 714/8	May 1989 N/A	**Crouse** et al. N/A
5212693 714/5	May 1993 **N/A**	Chao et al. N/A
5493674 395/182.04	February 1996 N/A	**Mizutani** et al. N/A
5546586 395/705	August 1996 **N/A**	Wetmore et al. N/A
5726641 340/825.22	March 1998 N/A	**Ide** N/A
5757690 365/104	May 1998 **N/A**	McMahon N/A
5796972 395/384	August 1998 N/A	**Johnson** et al. N/A
5802549 711/102	September 1998 **N/A**	Goyal et al. N/A
5829012 711/102	October 1998 N/A	**Marlan** et al. N/A

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO US-CL	PUBN-DATE	COUNTRY
0 049 353 A1	April 1982	EP
5-73292	March 1993	JP
WO 98/57255	December 1998	WO

OTHER PUBLICATIONS

Bradley et al., "Method of Customizing Patches for Each Hardware Configuration," IBM Technical Disclosure Bulletin, vol. 27, No. 4A, pp. 2187-88, Sep. 1984.

"Maintainable ROS Code Through the Combination of ROM and EEPROM," IBM Technical Disclosure Bulletin, vol. 32, No. 9A, pp. 273-76, Feb. 1990.

"Optimized Use of Code Patch Storage," IBM Technical

Disclosure Bulletin,
vol. 36, No. 5, pp. 469-70, pp. 469-70, May 1993.

Stevens, J. "DSPs in Communications," http://www.Spectrum.ieee.org/Spectrum/Sep. 98/features/dsp.html, 1998

ART-UNIT: 214

PRIMARY-EXAMINER: Donaghue; Larry D.

ATTY-AGENT-FIRM: Marshall, O'Toole, Gerstein, Murray & Borun

ABSTRACT:

A processing device includes a **ROM** having program instructions and at leas one jump instruction stored therein, a **patch** program for **patching** the program instructions in the **ROM, a RAM** capable of storing the **patch** program and a **patch** vector **table** that indicates the location of the **patch** program. A processor executes the program instructions in the program **ROM** and uses the **patch** vector table to execute the **patch** program when one of the jump instructions is reached. The patch program may be stored in the **RAM** and the patch vector **table** may point to the address in the **RAM** at which the patch program is stored so that the processor jumps directly to the **RAM** to execute the patch program when it reaches the one of the jump instructions. Likewise, a patch engine may be used to locate a specified patch program, load the specified patch program into the RAM and execute the specified patch program when the processor reaches the one of the jump instructions.

42 Claims, 4 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 4

---------- KWIC ---------

Abstract Text - ABTX (1):
A processing device includes a **ROM** having program instructions and at leas one jump instruction stored therein, a **patch** program for **patching** the program instructions in the **ROM, a RAM** capable of storing the **patch** program and a **patch** vector **table** that indicates the location of the **patch** program. A processor executes the program instructions in the program **ROM** and uses the **patch** vector table to execute the **patch** program when one of the jump instructions is reached. The patch program may be stored in the **RAM** and the patch vector **table** may point to the address in the **RAM** at which the patch program is stored so that the processor jumps directly to the **RAM** to execute the patch program when it reaches the one of the jump instructions. Likewise, a patch engine may be used to locate a specified patch program, load the specified patch program into the RAM and execute the specified patch program when the processor reaches the one of the jump instructions.

Brief Summary Text - BSTX (13):
A processing device having a **ROM** capable of being **patched** includes a **ROM** that stores program instructions and at least one jump instruction, a **patch** program, a **RAM** memory capable of storing the **patch** program, and a **patch** vector **table** that indicates the location of the **patch** program. A processor executes the program instructions in the program **ROM** and uses the **patch** vector table to execute the **patch** program when the processor reaches one of the jump

instructions. If desired, the patch program may be initially stored in the **RAM** and the patch vector **table** may point to the address in the **RAM** at which the patch program is stored. In this case, the processor jumps directly to the RAM to execute the patch program when it reaches a jump instruction at which a patch is to be performed. However, if no **patch** program is to be executed at a jump, the **patch** vector **table** points to an address within the **ROM** which causes the processor to jump directly to another address within the **ROM** without jumping into the **RAM**.

Drawing Description Text - DRTX (3):
FIG. 2 is a memory map for a program **ROM,** a program **RAM** and a data **RAM** of FIG. 1, illustrating a method of **patching ROM** code using a **patch** vector **table**;

Drawing Description Text - DRTX (5):
FIG. 4 is a memory map of the program **ROM,** the program **RAM** and the data **RAM** of FIG. 1, illustrating a method of **patching ROM** code using a **patch** engine and a **patch** engine data **table**.

Detailed Description Text - DETX (6):
The **patch** vector **table** may be stored in the external **ROM** 10 or the external **RAM** 15 and, if desired, may be transferred to the data **RAM** 65 of the ASIC 20 when **patching** is to be performed. However, the patch vector **table** may be stored at any other desired location within the processing device 5, such as in the program **RAM** 50, as long as the patch vector **table** is accessible by the slave processor 30.

Detailed Description Text - DETX (8):

Initially, patch programs, which may include one or more patch instructions, are stored in the external memory 7 and are loaded into the program **RAM** 50 preferably at the same time that the patch vector **table** is loaded into the data **RAM** 65. Thereafter, as the slave processor 30 executes the program instructions stored in program **ROM** 45, the slave processor 30 is directed, at each of the jump instructions, to jump to a location specified by the **patch** vector associated with that jump instruction. As indicated above, when no **patch** is to be performed at a jump, the **patch** vector for that jump specifies another address within the program **ROM** 45 and the processor 30 jumps directly to that address (which may be the next address, a subroutine or any other address within the program **ROM** 45) to continue execution of the code stored in the program **ROM** 45. However, when a patch is to be performed at a jump instruction, the patch vector for that jump instruction is an address (or some other value that points to an address) in the program RAM 50 where the appropriate patch program is stored. Upon reaching such a jump instruction, the processor 30 jumps to the patch program in the program RAM 50 and executes the patch program. The **patch** program includes a jump instruction at the end thereof which causes the processor 30 to jump back to any desired address within the program **ROM** 45. This **return address** may be the next address within the program **ROM** 45 (following the jump instruction that caused the **patch** program to be executed) when, for example, the **patch** program provides some enhanced function. Alternatively, the **return address** may be an address elsewhere within the program **ROM** 45 causing one or more of the original **ROM**

program instructions to be skipped when, for example, some of the code within
the program **ROM** 45 needs to be replaced by the **patch** program.

Detailed Description Text - DETX (9):
An example of this **patch** mechanism will be described with respect to FIG. 2,
which illustrates the relevant contents of the program **ROM** 45, the program RAM
50 and the data RAM 65 for a section of **ROM** code that is unpatched and for that
same section of **ROM** code when that code has been **patch with a patch** program.
As illustrated on the left side of FIG. 2, unpatched code instructions 1-3 and
10-16 and a jump instruction 67 are stored in the program ROM 45. The jump
instruction 67 refers to or points to a patch vector (which is an address or
other pointer) stored at an address N within, for example, the data RAM 65.
The **patch** vector at the address N is one entry of a **patch** vector **table** stored
in the data **RAM** 65, this **patch** vector **table** including an entry for each jump
instruction within the program **ROM** 45. In the example of FIG. 2, the **patch**
vector at the address N associated with the jump instruction 67 is the address
A. The **patch** vectors within the **patch** vector table for jump instructions not
shown in the program **ROM** 45 of FIG. 2 are illustrated as "Address XXX".
Because the **patch** vector at address N points back to address A within the
program **ROM** 45, no **patch** program is to be executed at the jump instruction 67.
As a result, no patch program is stored in the program RAM 50.

Detailed Description Text - DETX (11):
Referring now to the right side of FIG. 2, the patch vector table has been
configured to run a patch program at the jump instruction 67. In particular,

the patch vector at the address N within the patch vector **table** has been changed to address B, which is an address within the program **RAM** 50. Likewise, a **patch** program having instructions 20-22 and a jump to address C of the program **ROM** 45 is stored at the address B of the program RAM 50. Now, when run, the processor 30 executes instruction 1-3 in the program **ROM** 45 and, when it reaches the jump instruction 67, the processor 30 jumps to the address specified by the **patch** vector at the address N of the data RAM 65, namely address B. The processor 30 then executes the **patch** program instructions 20-22 starting at address B within the program RAM 50 and then jumps to the address C within the program **ROM** 45, as directed by the jump instruction immediately following the instruction 22. As will be understood, the **patched** instruction sequence is **ROM** instructions 1-3, **patch** program instructions 20-22 and **ROM** instructions 14-16.

Detailed Description Text - DETX (12):
It will be understood that the **patch** vector table may initially be stored in the data **ROM** 60 having default **patch** vectors to be used when no **patching** is needed, such default **patch** vectors assuring that the code within the program **ROM** 45 is implemented in a desired order. The default patch vector **table** may be transferred to the data **RAM** 65 upon power-up or at some other convenient time. When patching is to be used, a new or updated patch vector **table** may be loaded from, for example, the external memory 7, into the data **RAM** 65 and the updated patch vector **table** may cause one or more patch programs (which are loaded into the program **RAM** 50) to be executed at the appropriate time.

However, if desired, the patch vector **table** may be stored in the external
memory 7 and may be accessed by the processor 30 without storing the patch
vector **table** in the data **RAM** 65. Also, if desired, the patch vector **table** may
be stored in the program **RAM** 50, although this uses up precious space within
the program **RAM** 50. Furthermore, it will be understood that the jump
instructions within the program **ROM** 45 may be direct jump commands, may be
calls such as subroutine calls, or may be any other type of instruction that
causes redirection of the program based on the contents of the **patch** vector
table. Still further, it will be understood that the final instruction of each
patch program may cause a jump to any desired address within the program **ROM**
45.

Detailed Description Text - DETX (18):

If desired, each patch program stored in the data RAM 65 and loaded into the
cache 70 can have a jump instruction at the end thereof causing a direct jump
back to a desired address within the program ROM 45. In this instance, the
patch program stub can be a single patch engine call in the form of a jump
instruction that specifies a patch program for execution and that causes a jump
to the patch engine 75. Of course, the patch engine 75 loads the specified
patch program into the RAM cache 70 and jumps to the beginning of that program
instead of calling that program as a subroutine. Then, when the patch program
is completed, the jump at the end of the patch program causes a direct jump
back to the program ROM 45 thereby completing the patch. Of course, other
variations in the manner in which a program stub causes the patch engine 75 to
load and execute a patch program and then causes a **return to**

a desired address
within the program **ROM** 45 can be used as well.

Detailed Description Text - DETX (20):
In another embodiment, efficiency may be further enhanced through the use of a patch engine data **table** that includes data pertaining to each patch program stored in the data **RAM** 65. Such a **patch** engine data table is used by the **patch** engine 75, in conjunction with a stack, to find, load and execute a **patch** program and then to return to the **patch** program stub which called the **patch** engine or directly to the program **ROM** 45. In general, a patch engine call automatically causes a return address to be placed on a stack associated with the processor 30.

Detailed Description Text - DETX (21):
The patch engine 75 uses that return address to identify a patch program to be executed for that patch engine call and, in particular, pops the return address off of the stack and compares that return address to a number of patch program identifiers stored in a patch engine data table to thereby identify which patch program is to be loaded and executed for that patch engine call.
When the patch engine 75 finds a match for the return address within the patch engine data table, the patch engine 75 reads information associated with the matching patch program identifier to enable the patch engine 75 to locate, load and execute the appropriate patch program. This data may take the from of a patch program address indication specifying the location or address of the patch program to be run, a length indication indicating the number of instructions within the patch program to be run, and a return address

specifying the address to which the processor 30 should return when it has completed execution of the patch program. Of course, other information may be provided in the alternative or in addition to that described herein. The **patch** engine data **table** may be stored at any desired location including, for example, within the data **RAM** 65, the program **RAM** 50, or the **ROM** 10 or **RAM** 15 of the external memory 7.

Detailed Description Text - DETX (23):

Just as in the embodiment described with respect to FIG. 3, the processor 30 begins execution at the address S in the program ROM 45, where instructions 1-3 are executed. After instruction 3 is executed, the jump instruction 67 directs program flow to the contents of address B in the program RAM 50. The first instruction of the patch program stub at address B is a patch engine call that calls the patch engine 75 in program ROM 45. At this point, the return address for the patch engine call, namely address B+1, is automatically pushed onto the stack 90. The processor 30 then begins execution of the patch engine 75 which pops the address B+1 off of the stack 90 and compares that address to one or more of the patch program identifiers within the patch engine data table 85. When a match is found, the patch engine 75 uses the address indication at the specified entry in the patch engine data table (e.g., A1 in FIG. 4) to locate the address at which the patch program to be executed is stored. The patch engine 75 uses the length indication for the specified entry to determine the length of the patch program to be loaded into the RAM cache 70 and places the return address of the specified entry onto the top of the stack 90. In FIG. 4, the address indication for the program identifier "B+1" is address A1,

indicating that the specified patch program is stored at address A1. Likewise, the length indication for the program identifier "B+1" is "3", indicating that the patch program stating at the address A1 is three instructions long. In the configuration of FIG. 4, the patch engine 75 loads the three instructions of the patch program starting at the address Al into the RAM cache 70 and executes that patch program. Thereafter, when the patch engine returns to the calling routine, the processor 30 pops the address off the top of the stack (which is the return address specified by the patch engine data table) and returns to that address. If desired, the return address may simply be the address used to identify the specified patch program in the first place, namely B+1 in the example of FIG. 4. If this is the case, the instruction at the address B+1 (which is a jump to a desired address in the program ROM 45) is executed. However, if desired, the **return address** within the patch engine data table 85 may be an address within the program **ROM** 45 (such as the address C) causing the patch engine 75 to **return directly to the address** C within the program **ROM** 45 without returning to the patch program stub. This technique eliminates the need to return to various places in the code that merely direct code execution flow which, in turn, reduces the execution time of the patch program.

Detailed Description Text - DETX (24):

In an alternate embodiment, the **patch** engine 75 may be called directly by a jump instruction (e.g., jump instruction 67) within the program **ROM** 45 without the use of the **patch** vectors within the **patch** vector **table** or the need for **patch** engine calls within the program **RAM** 50. In this embodiment, the patch

engine 75 uses the contents of the top of the stack 90 (specifying the jump instruction which called the patch engine 75) and the patch engine data table 85 to locate a specified patch program for execution, in the manner described above. Of course, in this configuration, the **return address for the patch** program entry will specify an address within the program **ROM** 45. During operation of this configuration, the jump instruction 67 calls or otherwise jumps directly to the patch engine 75. The patch engine 75 then uses the address on the stack 90 and the patch engine data table 85 to identify and locate the patch program to be executed. If no **patch** program is to be executed, the **patch** program address or length indication may be a unique code that tells the **patch** engine 75 to simply return to the program **ROM at the return address** specified by the **patch** engine data table 85. If a **patch** is to be performed, however, the **patch** engine 75 finds, loads and executes the specified **patch** program based on the information within the **patch** engine data table 85 and then **returns to the address** within the program **ROM** 45 indicated by the **return address of the patch** engine data table 85. As noted above, this configuration further reduces the amount of space needed in the program RAM 50 as it eliminates the need to provide patch program stubs or patch engine calls within the program RAM 50. Likewise, this configuration streamlines execution of a patch program.

Claims Text - CLTX (19):

12. The processing device of claim 8, wherein the **patch** program identifier is indicative of an address within the program **RAM, wherein the patch** engine data **table** includes a number of entries and one of the

entries includes the
patch program identifier, the **address indication and a return address and**
wherein the patch engine compares an actual address associated with the first
portion of the **patch** program in the **RAM with the patch** program identifier to
locate a desired entry in the **patch** engine data **table,** uses the address
indication at the desired entry to locate the second portion of the **patch**
program in the further memory and uses the **return address to cause a return** to
the first portion of the **patch** program within the program **RAM** or to the program
instructions within the program **ROM**.

Claims Text - CLTX (41):

21. The processing device of claim 20, further including a **patch** vector
table having a patch vector for each one of the plurality of jump instructions,
wherein each of the **patch** vectors points to a **ROM** address within the **ROM or to**
a RAM address within the first section of the **RAM,** and wherein the processor
uses the **patch** vector **table** to perform a jump when it reaches any one of the
plurality of jump instructions.

Claims Text - CLTX (47):

27. The processing device of claim 24, wherein each of the **patch** program
identifiers is indicative of an address within the **RAM** associated with one of
the **patch** engine calls, wherein the **patch** engine data **table** includes a number
of entries, each entry having one of the **patch** program identifiers, one of the
address indications and a return address and wherein the
patch engine compares
an actual address associated with the **RAM** to one or more of the **patch** program
identifiers to locate a desired entry for the specified **patch**

program, uses the
address indication at the desired entry to locate the
specified **patch** program
and uses the **return address** at the desired entry to cause a
return to the **RAM**
or to the ROM.

Claims Text - CLTX (65):
39. The processing device of claim 37, wherein each of
the patch program
identifiers is indicative of an address within the **ROM,**
wherein the patch
engine data table includes a number of entries, each entry
including one of the
patch program identifiers, one of the **address indications and**
a return address
and wherein the patch engine compares an actual address
associated with the **ROM**
to one or more of the patch program identifiers to locate a
desired entry for
the specified patch program, uses the address indication at
the desired entry
to locate the specified patch program and uses the **return**
address at the
desired entry to cause a return to the **ROM**.